Chapitre 4

Algorithmes de traçage

4.1 Tracé d'une droite

Dans cette section nous étudions le traçage d'un segment d'équation $y = mx + b$ pour $x \in [x_1, x_2]$ avec $x_1 < x_2$ (le traçage d'un segment vertical est trivial et le cas $x_1 > x_2$ se traite en échangeant les valeurs de x_1 et x_2). Tracer un segment revient à déterminer par quels pixels celui-ci passe : c'est la `rasterization`.

4.1.1 Algorithme naïf

La méthode naïve consiste à calculer pour chaque pixel d'abscisse x son ordonnée à partir de l'équation :

```
pour x=x1 à x2 faire
     afficher(x, arrondir(m*x+p))
fin pour
```

Cette méthode nécessite cependant de nombreux calculs en nombres flottants (m étant un réel). On peut facilement éviter la multiplication grâce à l'algorithme qui suit.

4.1.2 Algorithme incrémental initial

Soient (x'_1, y'_1) et (x'_2, y'_2) deux points du segment, posons

$$\Delta x = x'_2 - x'_1, \qquad \Delta y = y'_2 - y'_1.$$

On a alors

$$\Delta y = m\Delta x.$$

Ainsi, si $0 \leqslant m < 1$, on peut tracer le segment en fixant $\Delta x = 1$ et alors $\Delta y = m$.

L'algorithme devient donc :

```
y=m*x1+b
x=x1
tant que  x<x2 faire
     afficher(x,arrondir(y))
```

```
    x=x+1
    y=y+m
fin tant que
```

On généralise facilement cet algorithme à toute valeur de m.

On a ainsi minimisé le nombre d'opérations à effectuer. La valeur de y doit cependant toujours être arrondie.

4.1.3 Algorithme de Bresenham

La dernière amélioration que nous présentons permet de ne réaliser que des opérations sur les entiers.

Considérons le cas précédent : $0 \leqslant m < 1$ (appelé cas du premier octant, *cf.* Fig. 4.1).

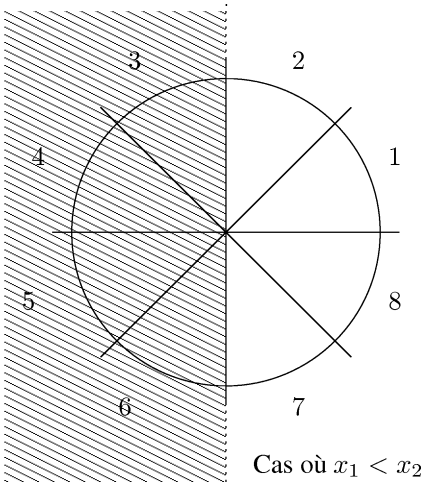

FIG. 4.1 – *Découpage du plan en octant*

Losqu'on incrémente la valeur de x, il faut alors décider s'il faut allumer le pixel $(x+1, y+1)$ où le pixel $(x+1, y)$.

Le principe de l'algorithme de Bresenham est de considérer le segment comme une suite de carrés de côtés 1 dont le coin inférieur gauche est d'abscisse entière et sur le segment. On allume alors le pixel qui est le plus couvert par ce carré (Fig. 4.2).

Soit (x_i, y_i) le pixel allumé à l'étape i. Il faut alors calculer (x_{i+1}, y_{i+1}). On a $x_{i+1} = x_i + 1$. La droite "réelle" passe donc par le point $y = m(x_i + 1) + b$. Calculons les valeurs d_1, d_2 définies comme suit (Fig. 4.3) :

$$\begin{array}{rcccl} d_1 & = & y - y_i & = & m(x_i + 1) + b - y_i, \\ d_2 & = & (y_i + 1) - y & = & y_i + 1 - m(x_i + 1) - b \end{array}$$

Ainsi, d_1 correspond à l'aire commune au carré de coté 1 et au pixel (x_{i+1}, y_{i+1}) et d_2 correspond à l'aire commune au carré de coté 1 et au pixel (x_{i+1}, y_i).

L'algorithme de Bresenham décide alors du pixel à allumer selon le signe de $d_1 - d_2$:

– Si $d_1 - d_2 < 0$ alors on affiche le pixel $(x_i + 1, y_i)$,

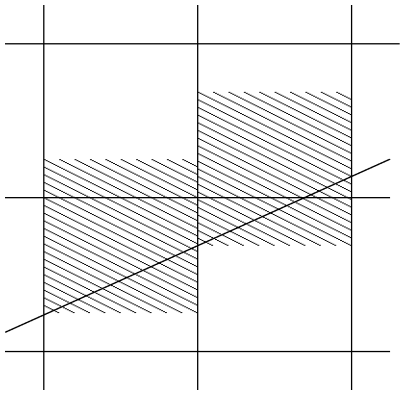

FIG. 4.2 – *Rasterization par l'algorithme de Bresenham*

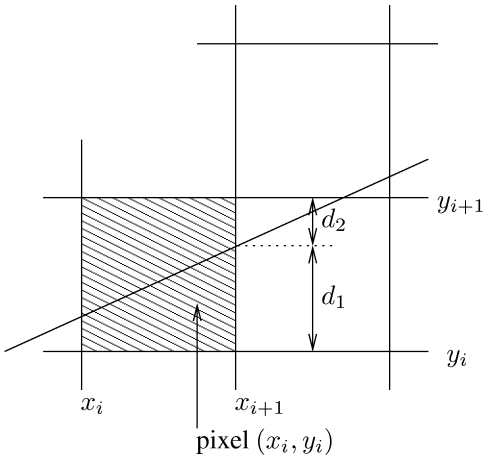

FIG. 4.3 – *Droite se situant dans le premier octant*

– Sinon on affiche le pixel $(x_i + 1, y_i + 1)$.

Montrons que cette algorithme peut-être écrit en n'utilisant que des operations sur les entiers.

On a

$$\begin{aligned}(d_1 - d_2) &= m(x_i + 1) + b - y_i - (y_i + 1 - m(x_i + 1) - b) \\ &= 2m(x_i + 1) + 2b - 2y_i - 1\end{aligned}$$

Or on a $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$, d'où

$$(d_1 - d_2) = 2\frac{\Delta y}{\Delta x}(x_i + 1) + 2b - 2y_i - 1$$

Posons

$$p_k = 2\Delta y x_k - 2\Delta x y_k + c \text{ où } c = 2\Delta x b - \Delta x + 2\Delta y.$$

On a alors

$$p_{i+1} - p_i = 2\Delta y(x_{i+1} - x_i) - 2\Delta x(y_{i+1} - y_i)$$

Comme $x_{i+1} = x_i + 1$, on a

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i)$$

et

$$\begin{aligned} p_1 &= 2\Delta y x_1 - 2\Delta x y_1 + 2\Delta x b - \Delta x + 2\Delta y \\ &= 2\Delta y x_1 - 2\Delta x y_1 + 2\Delta x (y_1 - \frac{\Delta y}{\Delta x} x_1) - \Delta x + 2\Delta y \\ &= 2\Delta y - \Delta x \end{aligned}$$

Enfin, on a

$$p_i = 2\Delta y x_i - 2\Delta x y_i + 2\Delta x b - \Delta x + 2\Delta y$$

et

$$(d_1 - d_2) = 2\frac{\Delta y}{\Delta x}(x_i + 1) + 2b - 2y_i - 1$$

d'où

$$p_i = \Delta x (d_1 - d_2).$$

On a pris $x_1 > x_2$, donc $\Delta x > 0$. Ainsi p_i et $d_1 - d_2$ sont de même signe.

L'algorithme de Bresenham s'écrit donc :

```
Delta_x=x2-x1
Delta_y=y2-y1
pi=2*Delta_y-Delta_x
xi=x1
yi=y1
tant que xi<x2 faire
     afficher(xi,yi)
     xi=xi+1
     si pi>=0 alors yi=yi+1
     si pi<0 alors
        pi=pi+2*Delta_y
     sinon
        pi=pi+2*(Delta_y-Delta_x)
     fin si
fin tant que
```

Cet algorithme se généralise facilement aux huit octants.

4.1.4 Autres critères

Il existe d'autres algorithmes de tracé de segment utilisant d'autres critères. En particulier, OpenGL recommande d'utiliser le critère du «diamond-exit» (*cf.* Fig. 4.4). Un pixel (x, y) sera alors allumé si le segment traverse le losange $R_{(x,y)}$ contenu dans le pixel :

$$R_{(x,y)} = \left\{ (x', y') \;\middle|\; \left| x' - x - \frac{1}{2} \right| + \left| y' - y - \frac{1}{2} \right| < \frac{1}{2} \right\}.$$

On montre que lorsque les extrémités du segment reposent au centre de deux pixels alors ce critère amène à dessiner les mêmes pixels que l'algorithme de Bresenham.

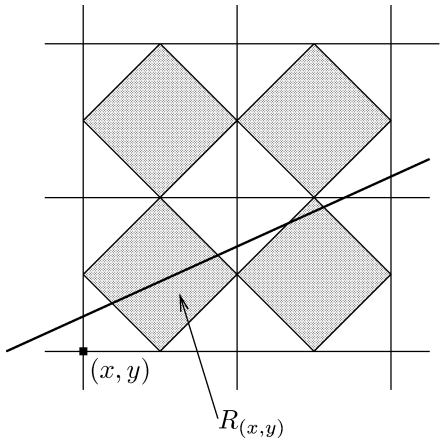

FIG. 4.4 – *Le critère du «diamond-exit».*

4.2 Aliasing

4.2.1 Le problème de l'aliasing

Le problème de l'aliasing est à l'origine un problème d'échantillonnage de signal. Afin d'obtenir un échantillon fidèle, on doit respecter le théorème de Shannon :

Théorème 4.2.1 (Shannon) *La fréquence d'échantillonnage doit être égale ou supérieure au double de la fréquence de la plus haute composante spectrale du signal.*

En effet, si on ne respecte pas ce théorème on ne pourra pas alors reconstituer de façon satisfaisante le signal d'origine à partir de son échantillon (Fig. 4.5). Un signal de même échantillonnage que le signal d'origine est appelé un *alias*.

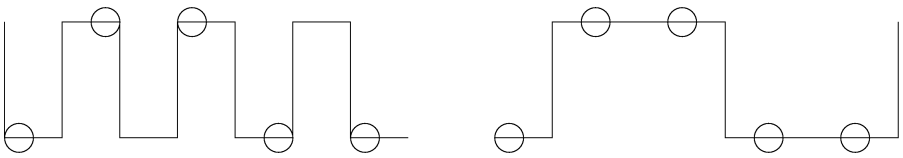

FIG. 4.5 – *Deux signaux de même échantillonnage*

La rasterization revenant à échantillonner, on doit se conformer au théorème de Shannon. Il existe deux moyens pour combattre le problème de l'aliasing :

– Augmenter la fréquence d'échantillonage (ce qui revient à augmenter la résolution de l'écran).
– Moyenner sur l'intervalle d'échantillonage.

La première méthode étant souvent inemployable, nous présentons dans la section suivante une amélioration de l'algorithme de Bresenham reposant sur le moyennage.

4.2.2 Algorithme de Bresenham avec anti-aliasing

Au lieu de dessiner un seul pixel à chaque abscisse, on va dessiner deux pixels en affectant une luminosité tenant compte de l'erreur induite par la rasterization (Fig. 4.6).

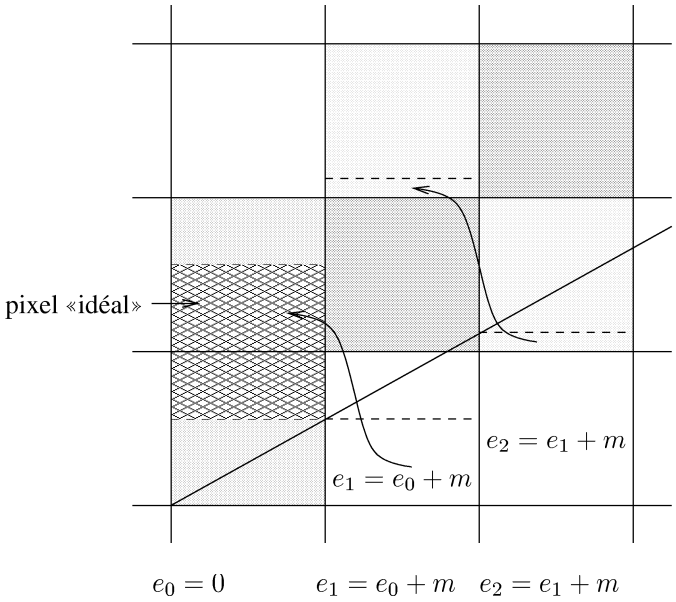

FIG. 4.6 – *Algorithme de Bresenham avec anti-aliasing*

Voici l'algorithme de Bresenham avec anti-aliasing :

```
y=y1
e=0
pour x=x1 à x2 faire
     e=e+m
     si e>=1 alors
        e=e-1
        y=y+1
     fin si
     afficher(x,y+1,e) // e représente ici la luminosité
     afficher(x,y,1-e)
fin pour
```

La ligne tracée aura alors des contours plus «doux». On peut vérifier que cet algorithme n'affiche pas nécessairement les pixels allumés par l'algorithme de Bresenham (sans anti-aliasing). Nous laissons au lecteur le soin d'écrire une version qui allumerait ces pixels.

4.3 Tracé d'un cercle

Il existe bien sûr des algorithmes qui permettent de tracer des primitives autres que des segments.

Nous proposons, en guise d'exemple, un algorithme permettant de tracer un cercle (algorithme de Michener ou algorithme de Bresenham pour le cercle).

Nous nous concentrerons sur le tracé de l'octant Nord-Nord-Est d'un cercle de rayon R, centré en O. Nous traçons en partant du point $(0, R)$. Ainsi lorsqu'un point est tracé, il faut décider si on doit allumer le point «Est» ou «Sud-Est» (Fig. 4.7).

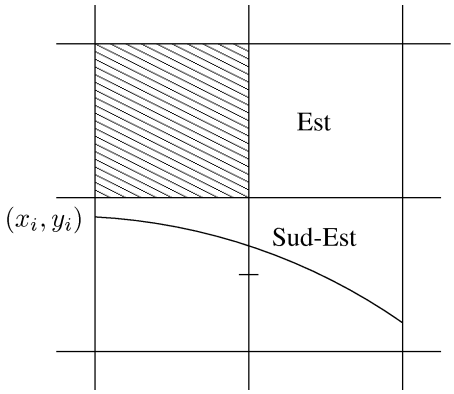

FIG. 4.7 – *Traçage d'un cercle*

Le point (x_i, y_i) étant tracé, pour décider si on doit allumer le pixel $(i+1, y_i)$ ou le pixel $(i+1, y_i-1)$ on va comparer le rayon du cercle centré en O et passant par le point $(i+1, y_i - \frac{1}{2})$ à R. Si ce rayon est plus grand que R, on affiche le point $(i+1, y_i-1)$ et sinon on affiche le point $(i+1, y_i)$.

Le rayon du cercle passant par $(i+1, y_i - \frac{1}{2})$ est

$$R_{i+1,y_i-\frac{1}{2}} = \sqrt{(i+1)^2 + \left(y_i - \frac{1}{2}\right)^2}.$$

On doit donc regarder le signe de $R_{i+1,y_i-\frac{1}{2}} - R$, ce qui revient à regarder le signe de $E_{i+1} = R^2_{i+1,y_i-\frac{1}{2}} - R^2$ (on suppose que les rayons sont plus grands que 1). Si c'est négatif, on allume le point $(i+1, y_i)$ et sinon on allume le point $(i+1, y_i-1)$.

Ce qui donne pour un rayon entier :

- En $(0, R)$, on a allume le pixel $(0, R)$.
- Pour le point d'abscisse 1, on a $E_1 = 1 + (R - \frac{1}{2})^2 - R^2 = \frac{5}{4} - R$.
- Pour le point d'abscisse $i+1$,
  - Si on a allumé le pixel «Est» à l'étape i, alors $y_i = y_{i-1}$, et donc

$$\begin{aligned} E_{i+1} &= (i+1)^2 + \left(y_i - \frac{1}{2}\right)^2 - R^2 \\ &= 2i + 1 + \left(i^2 + \left(y_{i-1} - \frac{1}{2}\right)^2 - R^2\right) \\ &= 2i + 1 + E_i \end{aligned}$$

  - Sinon, on a $y_i = y_{i-1} - 1$, et donc

$$\begin{aligned} E_{i+1} &= (i+1)^2 + \left(y_i - \frac{1}{2}\right)^2 - R^2 \\ &= i^2 + 2i + 1 + \left(y_{i-1} - \frac{3}{2}\right)^2 - R^2 \end{aligned}$$

$$
\begin{aligned}
&= i^2 + 2i + 1 + y_{i-1}^2 + \frac{9}{4} - 3y_{i-1} - R^2 \\
&= 2i + 1 + \frac{8}{4} - 2y_{i-1} + \left(i^2 + \left(y_{i-1} - \frac{1}{2} \right)^2 - R^2 \right) \\
&= 2i - 2y_{i-1} + 3 + E_i \\
&= 2i - 2y_i + 1 + E_i
\end{aligned}
$$

On peut, bien sûr, généraliser cette algorithme à tout rayon et à tous les octants. De plus, il est possible d'écrire une version de cet algorithme n'utilisant que des opérations sur les entiers.

4.4 Remplissage de rectangles

Nous présentons dans cette section un bref descriptif des techniques mises en œuvre pour tracer des surfaces pleines.

4.4.1 Rectangles pleins

Tracer un rectangle plein revient à tracer une succession des segments horizontaux.

Afin d'éviter les problèmes de recouvrements en cas de juxtaposition (fig 4.8), on ne trace pas une frontière si le demi-plan inférieur (resp. gauche) contient le rectangle (fig 4.9).

4.4.2 Polygônes pleins

L'algorithme de remplissage d'un polygône peut s'écrire ainsi (fig. 4.10) :

- Recherche des points d'intersection d'une ligne horizontale avec des contours (en nombre pair)
- Remplissage de la ligne par règle de parité

On prend les points intérieurs au polygône afin d'éviter les chevauchements.

On remplit en utilisant des listes chaînées des arêtes classées par ordonnées minimales décroissantes et une liste chaînée unique des arêtes actives.

Si la frontière n'est pas horizontale, on conserve le sommet inférieur (fig.4.11).

Le calcul des intersections avec les frontières se fait sur des nombres entiers. En effet, considérons une arête d'extrémités (x_{bas}, y_{bas}), (x_{haut}, y_{haut}) avec $y_{bas} < y_{haut}$ (on ne tient pas compte des arêtes horizontales).

Lors du traçage par « scan-line », on fait varier les ordonnées de un en un, on a donc :

$$
\left\{
\begin{array}{rcl}
x_{i+1} & = & x_i + \frac{1}{m} \\
y_{i+1} & = & y_i + 1
\end{array}
\right.
$$

où m est la pente, et donc

$$
\frac{1}{m} = \frac{x_{haut} - x_{bas}}{y_{haut} - y_{bas}}
$$

À chaque étape nous devrons allumer le pixel $(\lfloor x_i \rfloor, y_i)$.

Posons

$$
\forall k, \quad x_k = \lfloor x_k \rfloor + \frac{r_k}{y_{haut} - y_{bas}}
$$

On a $x_{i+1} = x_i + \frac{1}{m}$ et donc

$$(y_{haut} - y_{bas})\lfloor x_{i+1} \rfloor + r_{i+1} = (y_{haut} - y_{bas})\lfloor x_i \rfloor + r_i + (x_{haut} - x_{bas})$$

L'algorithme de calcul des intersection peut donc être écrit en utilisant une variable `x` pour mémoriser $\lfloor x_i \rfloor$ et une variable `R` pour mémoriser r_i. Alors, pour passer de l'étape i à l'étape $i+1$, on a

– Si $r_i + (x_{haut} - x_{bas}) \geqslant y_{haut} - y_{bas}$ alors

$$\lfloor x_{i+1} \rfloor = \lfloor x_i \rfloor + \left\lfloor \frac{r_i + (x_{haut} - x_{bas})}{y_{haut} - y_{bas}} \right\rfloor$$

et

$$r_{i+1} = r_i + (x_{haut} - x_{bas}) - (y_{haut} - y_{bas}) \left\lfloor \frac{r_i + (x_{haut} - x_{bas})}{y_{haut} - y_{bas}} \right\rfloor$$

– Sinon

$$\lfloor x_{i+1} \rfloor = \lfloor x_i \rfloor$$

et

$$r_{i+1} = r_i + (x_{haut} - x_{bas})$$

Ainsi les seules opérations qui interviennent sont des opérations sur des entiers.

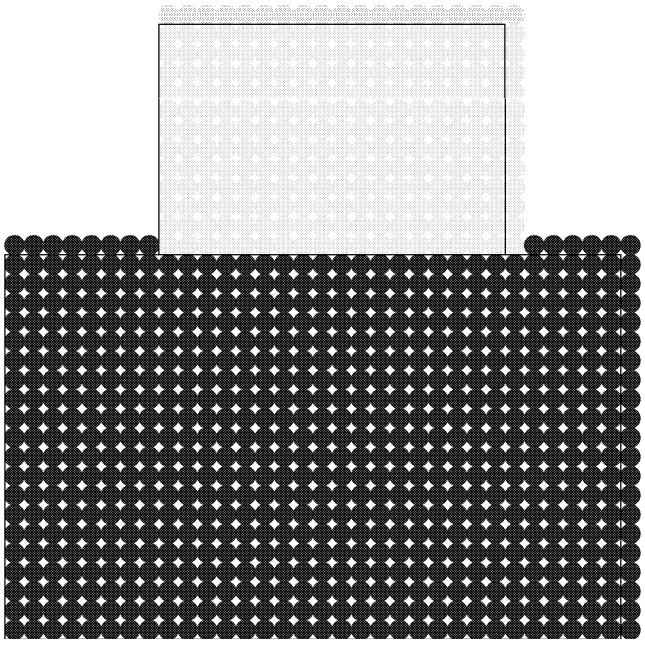

FIG. 4.8 – *Juxtaposition de rectangles pleins*

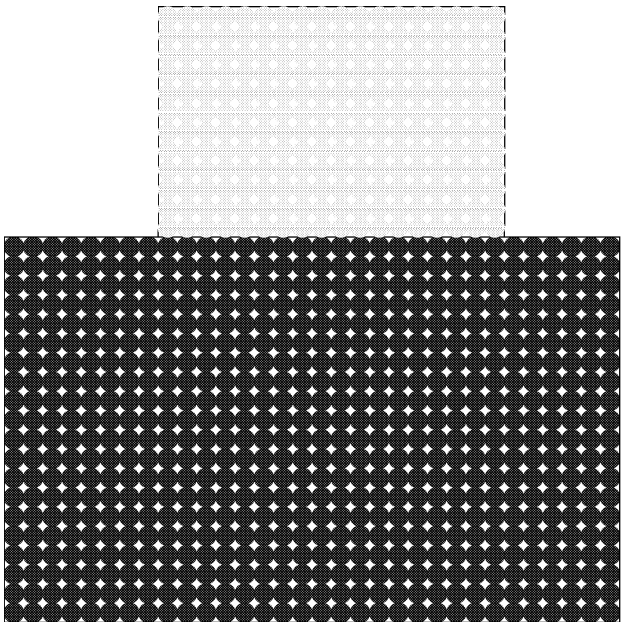

FIG. 4.9 – *Remplissage de rectangles pleins*

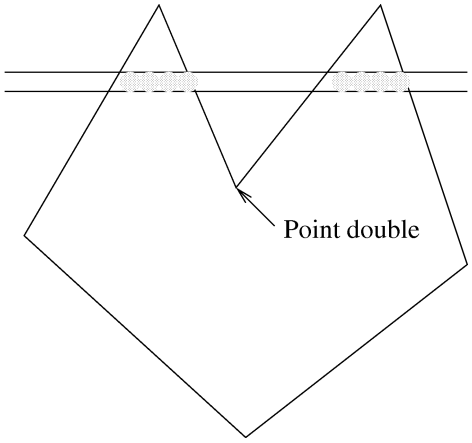

FIG. 4.10 – *Remplissage de polygônes*

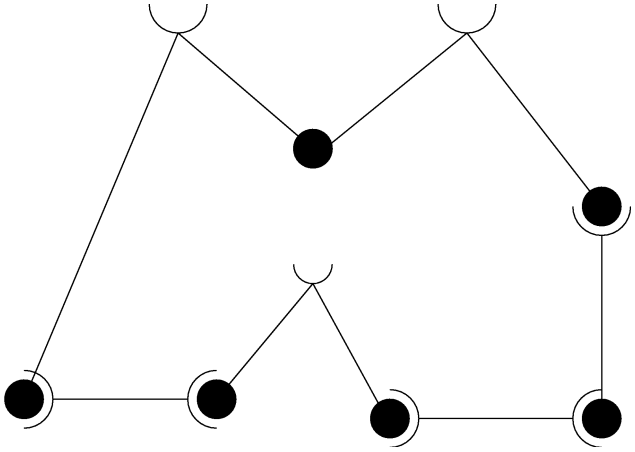

FIG. 4.11 – *Sommets conservés pour les arêtes*